# More on speeding up circularity tests for attribute grammars

Martin Jourdan, Didier Parigot

INRIA

UNITÉ DE RECHERCHE
INRIA-ROCQUENCOURT

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P.105
78153 Le Chesnay Cedex
France

Tél.:(1)39635511

Rapports de Recherche

N° 828

# MORE ON SPEEDING UP CIRCULARITY TESTS FOR ATTRIBUTE GRAMMARS

Martin JOURDAN
Didier PARIGOT

AVRIL 1988

*RR 828*

# More on Speeding Up Circularity Tests for Attribute Grammars

# Du Nouveau sur l'Accélération des Tests de Circularité pour les Grammaires Attribuées

*Martin Jourdan and Didier Parigot*

INRIA, Domaine de Voluceau, Rocquencourt, BP 105, F-78153 Le Chesnay, France

**Abstract** : Practical implementation of circularity tests for attribute grammars was the subject of many earlier papers, among which one by Deransart, Jourdan and Lorho [DJL 84], which presented three improvements to the basic algorithm by Knuth [Knu 71] : covering, partitioning and weak stability. These improvements reduced *practical* time and space complexity, allowing those theoretically exponential tests to be actually feasible. This paper presents two new improvements to these tests : semantic stability avoids recomputations which are a priori redundant, and thus useless ; non-terminals static ordering supplements partitioning by determining a better order for processing non-terminals inside each subgrammar, allowing information to propagate faster. These new improvements allow a gain on execution time by a factor of up to 3.5 for meaningfully large grammars ; however they do not change the space complexity.

**Keywords :** attribute grammars, circularity tests, graph ordering.

**Résumé** : L'implantation pratique des tests de circularité pour les grammaires attribuées a été dans le passé le sujet de nombreuses communications, en particulier une de Deransart, Jourdan et Lorho [DJL 84], qui présentait trois améliorations à l'algorithme de base de Knuth [Knu 71] : le "covering", le partitionnement et la stabilité faible. Ces améliorations réduisaient la complexité *pratique* en temps et en place, permettant à ces tests théoriquement exponentiels d'être en réalité faisables. Ce rapport présente deux nouvelles améliorations à ces tests : la stabilité sémantique évite des recalculs à priori redondants, et donc inutiles ; l'ordonnancement statique des non-terminaux complète le partitionnement en déterminant un meilleur ordre de traitement des non-terminaux dans chaque sous-grammaire, ce qui permet de propager plus vite l'information. Ces nouvelles améliorations permettent, sur des grammaires significatives, de gagner jusqu'à un facteur de 3,5 sur le temps d'exécution ; cependant elles ne changent pas la complexité en place.

**Mots clés :** grammaires attribuées, tests de circularité, ordonnancement de graphe.

# 1. Introduction

Since Knuth's original paper [Knu 68] about attribute grammars, one of the most debated topics was testing them for circularity (12 references in the most recent bibliography [DJL 85]). It was rapidly proved that this property is attached to the attribute grammar itself [Knu 71], and that the complexity of the test is intrinsically exponential [Jaz 81]. Since the latter property might prevent a practical use of attribute grammars, in spite of their other advantages, many authors worked out a number of improvements to reduce the practical time and space complexity of the circularity test [LP 75, Che 81, RS 82, DJL 84].

The latter paper by Deransart, Jourdan and Lorho presented three such improvements among the most efficient ones :

- *covering*, originally introduced by Lorho and Pair [LP 75], discards from the set of graphs attached to a non-terminal symbol those graphs which are included in ("covered by") others of the same set, thus reducing space consumption ; it also reduced time consumption because the internal loop of the algorithm [Knu 71, DJL 84] must be repeated for each combination of such graphs (see algorithm A2 in section 2) ;

- *partitioning*, originally introduced by Chebotar [Che 81], uses the syntactic dependencies between non-terminal symbols to split the original grammar into subgrammars, and establishes an optimal order for applying the test to each of them ; in this way, time complexity is reduced because convergence is reached faster on those subgrammars ; space consumption is also reduced because, after processing a subgrammar, you may discard the relations attached to the non-terminals which will no longer be referenced (algorithm A3P in section 2) ;

- *weak stability* allows to skip useless recomputations of relations on productions and/or non-terminals which generate only finite trees, thus saving some time (algorithm A3S in [DJL 84]).

These three methods can be used separately or together (algorithm A4 in [DJL 84] and in this paper). Their efficiency on meaningful attribute grammars is measured in [DJL 84], where the practical complexity of circularity tests is also discussed.

The present paper is a complement to [DJL 84] and presents two new improvements for circularity tests. Both of them aim at determining a more efficient order for processing non-terminals and productions *inside* each subgrammar, order which was left unspecified by Chebotar [Che 81]. The first improvement, *semantic stability*, dynamically determines this ordering by skipping the elementary step of the algorithm (see section 2) when we know, by checking the "ages" of the input data of this step, that it will be redundant and bring no new information. The second improvement, *non-terminals static ordering*, uses the syntactic dependencies between non-terminals in each subgrammar, and the fact that the computation of dependency relations proceeds bottom-up in the grammar, to establish a near-optimal order for processing the non-terminals, so that information propagates as fast as possible and that the number of iterations needed to reach convergence is minimal. The idea is to order those non-terminals from the *output points* of a subgrammar, which are "low" in the tree, to its *entry points*, which are "higher".

The sequel of this paper is organized as follows : section 2 will briefly recall some notations and results from [DJL 84], including some of their algorithms. The main part of the paper is sections 3 and 4, which respectively present semantic stability and non-terminals static ordering. Section 5 will give numerical figures expressing the results of our improvements on practical attribute grammars. It will be followed by some concluding remarks and the list of references.

In order to make the paper more easily readable, we present here a table of the names of the algorithms which are used in the sequel :

| | |
|---|---|
| A1 | standard algorithm [Knu 71] |
| A2 | A1 with covering [LP 75, DJL 84] |
| A3P | A2 with partitioning [Che 81, DJL 84] ; uses A2 as a sub-algorithm |
| A3S | A2 with weak stability [DJL 84] |
| A4 | algorithm with covering, partitioning and weak stability [DJL 84] ; uses A3S as a sub-algorithm |
| A2SS | A2 with semantic stability (section 3) |
| A5 | algorithm with covering, partitioning, weak stability and semantic stability ; uses A2SS as a sub-algorithm (section 3) |
| A2NTO | A2 with non-terminals static ordering (section 4) |
| A6 | algorithm with covering, partitioning, weak stability and non-terminals static ordering ; uses A2NTO as a sub-algorithm (section 4) |
| A7 | algorithm with covering, partitioning, weak stability, semantic stability and non-terminals static ordering ; uses a combination of A2SS and A2NTO as a sub-algorithm (section 5) |

SNC algorithm checking strong (absolute) non-circularity in polynomial time, including all of the above improvements plus a slight one described by Parigot [Par 85]

## 2. Notations and Recalls

This section recalls some of the notations, definitions and results from [DJL 84]. We'll try to make it as brief as possible while keeping it self-contained : more details can be found in the original paper.

### Notations and Definitions

An attribute grammar $AG$ is composed of :

- a context-free grammar $G = (N, T, P, Z)$, where $N$ and $T$ are the sets of non-terminal and terminal symbols, $P$ the set of productions, which have the form
$$p : X_0 \rightarrow X_1 \; X_2 \; ... \; X_{n_p}$$
where the terminals are discarded, and $Z$ the start symbol.

- a set $A$ of attributes, partitioned into inherited attributes $I$ and synthesized attributes $S$ ; each non-terminal $X \in N$ has a subset of $A$, $I(X)$ and $S(X)$, attached to it, except that $Z$ has no inherited attributes ;

- for each production $p$, a set of semantic rules defining the computation of the elements of $S(X_0)$ and $I(X_j)$ for $1 \leq i \leq n_p$, in term of the elements of $A(X_i)$, $0 \leq i \leq n_p$ ; the latter set is $AS(p)$, the set of all attributes occurrences of $p$.

$DS(p, a, i) \subset AS(p)$ gathers all the attribute occurrences of $p$ which are arguments of the rule computing $(a, i)$ (Dependency Set). The dependency relation of $p$, $DO(p)$, is the binary relation on $AS(p)$ representing the dependencies between attribute occurrences of $p$ :
$$(a, k)\, DO(p)\, (b, l) \Leftrightarrow (b, l) \in DS(p, a, k)$$
For any binary relation $d$ on a set $A$, $d^+$ denotes the transitive closure of $d$ on $A$.

Let $t$ be a derivation tree for $G$. The compound dependency graph of $t$, $DG(t)$, is obtained by pasting together the graphs $DO(p)$ according to the structure of $t$. The attribute grammar $AG$ is non-circular iff $DG(t)$ is acyclic for any derivation tree $t$ of $G$. Knuth [Knu 68] shows that this dynamic property is decidable and can be tested statically. The corresponding algorithm is presented e.g. in [DJL 84], where it is named A1. The time complexity of this test is intrinsically exponential [JOR 75]. ◊

## Covering

The first improvement presented in [DJL 84], originally introduced in [LP 75], replaces the set of dependency relations attached to each non-terminal, and constructed in the course of the algorithm, by the *covering* of this set, i.e. the set of maximal elements w.r.t. inclusion. This replacement is equivalent, in the sense that if a relation in a set of relations induces a circularity, then there exists a relation in the covering of this set which induces a circularity. However, the number of elements in a covering is much smaller than the one of the original set, so the new algorithm is much faster and uses less space. Here is this algorithm, in which $D(X)$ is the set of dependency graphs attached to non-terminal X :

```
Algorithm A2 (G : grammar) is :
1       for each X in N do D(X) = Ø
2 main : repeat
                convergence = true
3 outer :       for each X0 in N do
4                       for each p in P such that p : X0 → X1 X2 ... Xnp do
5 inner :                       for each (d1, d2, ..., dnp) ∈D(X1) × D(X2) × ... × D(Xnp) do
6                                       let d = DO(p) ∪ ⋃_{j=1}^{np} j.dj
7                                       compute d+
8                                       if d+ is circular then          AG is circular
                                                                        stop
                                        endif
9                                       let d0 be the restriction of d+ to A(X0)
10 increase :                           if d0 is not covered by D(X0) then
                                                d(X0) = covering (D(X0) ∪ {d0})
                                                convergence = false
                                        endif
11                              endfor inner
12                      endfor
13              endfor outer
14      until convergence.
```

In line 6 we mean $(a, j)\, j.d_j\, (b, j) \Leftrightarrow a\, d_j\, b$

## Chebotar's partitioning

The basic point of this improvement is to find an ordering of the non-terminals in the grammar such that the above algorithm can be applied to a sequence of subgrammars, keeping for the next stage only the results of the current stage. Any ordering may be used, but it is natural to take into account the "syntactic dependencies" between the non-terminals, expressed by the relation $\Gamma$ :

$$p : X_0 \rightarrow X_1 X_2 ... X_{n_p} \in P \Rightarrow X_0 \,\Gamma\, X_i,\ 1 \leq i \leq n_p$$

Connected components (equivalence classes) of $\Gamma$ are the subgrammars we are interested in. Then, relation $\Gamma_0$ represents the dependencies between connected components of $\Gamma$, inferred from $\Gamma$. It is interesting to distinguish *entry points* of these subgrammars B :

$$\forall X \in B,\ X \in EP(B) \Leftrightarrow \exists Y \notin B, Y\ \Gamma X.$$

An inverse topological sort of $\Gamma_0$ gives the order in which the subgrammars are to be processed ; it is an inverse sort because the set of relations attached to non-terminals are computed "bottom-up" in the grammar.

This order for processing subgrammars is used to assign a *priority* to each non-terminal, which is the number of the stage after which we can discard its relations. Entry points of a subgrammar will be assigned the priority of the last subgrammar which references them ; other non-terminals will be assigned the priority of their own subgrammar. Denoting by $k$ the number of subgrammars and by $B^i$ the subgrammar with rank $i$, the resulting algorithm is then :

**Algorithm A3P** ($G$ : grammar) **is :**

1 **build** the graphs $\Gamma$ and $\Gamma_0$ from $G$
2 **apply** algorithms for Sorting, Entry Points Priority, Other Symbols Priority.
3 **let** $i = 0$
4 **repeat**
5 **let** $i = i + 1$
6 **apply** algorithm $A_2(B^i)$ (to the sub-grammar $B^i$ )
7 **discard** the relations of non-terminals of priority $i$
8 **until** $i = k$.

*Example 1 [Che 81] :*

**Grammar :**

| | | | |
|---|---|---|---|
| $A_1 \rightarrow A_2$ | $A_2 \rightarrow A_3 \mid A_4$ | $A_3 \rightarrow A_2 A_5$ | $A_4 \rightarrow A_6 \mid A_7$ |
| $A_5 \rightarrow A_8$ | $A_6 \rightarrow A_9$ | $A_7 \rightarrow A_{10} \mid A_{11}$ | $A_8 \rightarrow A_{12}$ |
| $A_9 \rightarrow A_{13}$ | $A_{10} \rightarrow A_{14}$ | $A_{11} \rightarrow A_{15}$ | $A_{12} \rightarrow A_8$ |
| $A_{13} \rightarrow A_6$ | $A_{14} \rightarrow A_{15}$ | $A_{15} \rightarrow A_{16} \mid A_5$ | $A_{16} \rightarrow A_7$ |

**Equivalence classes and entry points**

| | |
|---|---|
| $B_1 = \{A_1\}$, | $EP_1 = \{A_1\}$ |
| $B_2 = \{A_2, A_3\}$, | $EP_2 = \{A_2\}$ |
| $B_3 = \{A_4\}$, | $EP_3 = \{A_4\}$ |
| $B_4 = \{A_5\}$, | $EP_4 = \{A_5\}$ |
| $B_5 = \{A_6, A_9, A_{13}\}$, | $EP_5 = \{A_6\}$ |
| $B_6 = \{A_7, A_{10}, A_{14}, A_{15}, A_{11}, A_{16}\}$, | $EP_6 = \{A_7\}$ |
| $B_7 = \{A_8, A_{12}\}$, | $EP_7 = \{A_8\}$ |

**Subgrammars ordering**

| Rank | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| Nodes | $B_5$ | $B_7$ | $B_4$ | $B_6$ | $B_3$ | $B_2$ | $B_1$ |

**Entry points priority**

| Priority | 7 | 6 | 5 | 3 |
|---|---|---|---|---|
| Nodes | $A_2$ | $A_4, A_5$ | $A_6, A_7$ | $A_8$ |

**Other Symbols priority**

| Priority | 1 | 2 | 4 | 6 | 7 |
|---|---|---|---|---|---|
| Nodes | $A_9, A_{13}$ | $A_{12}$ | $A_{10}, A_{14}, A_{15}, A_{11}, A_{16}$ | $A_3$ | $A_1$ |

**Total Symbol priority**

| Priority | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| Nodes | $A_9, A_{13}$ | $A_{12}$ | $A_8$ | $A_{10}, A_{14}, A_{15}, A_{11}, A_{16}$ |
| Priority | 5 | 6 | | 7 |
| Nodes | $A_6, A_7$ | $A_3, A_4, A_5$ | | $A_2, A_1$ |

### Weak stability

Weak stability allows to skip recomputations of relations for non-terminals and/or productions which generate only finite trees of height $i$ after iteration $i$ (loop labelled "main" in A2). This is done as follows :

- at the beginning of the algorithm no non-terminal or production is weakly stable ;
- after each iteration :
  - mark as weakly stable those productions which have only weakly stable non-terminals in their right-hand side ;
  - mark as weakly stable those non-terminals which derive only weakly stable productions ;
- in the course of an iteration, do not take into account weakly stable non-terminals (line 3) and productions (line 4).

### Combination

All these improvements may be used separately or together : they are "orthogonal" to each other. However their combination is more efficient than each of them separately. The resulting algorithm, including covering, Chebotar's partitioning and weak stability, will be named A4.

Also notice that these improvements may be applied to other algorithms processing grammars and attribute grammars.

This ends the summary of the results presented in [DJL 84]. The sequel of the paper presents our new results.

## 3. Semantic Stability

This improvement aims at saving execution time in circularity tests by drastically reducing the number of redundant recomputations. It was introduced independently by Parigot [Par 85] and Chebotar [Che 85].

Let us turn back to the standard algorithm (A1 in [DJL 84]) or the one using covering (A2 in previous section). The inner loop of the algorithm (lines 5-11) chooses a combination of graphs in the set of relations attached to each non-terminal in the right-hand side of a production, builds the composite dependency graph, computes its transitive closure, tests its non-circularity, computes its projection onto the left-hand side non-terminal and, if necessary, adds the latter to the set of relations attached to that LHS non-terminal. This basic step must be repeated for each such combination, until convergence is reached.

The improvement discussed here, which we call *semantic stability*, is based on the observation that, if a given combination of graphs is "old" enough to have already been processed during a previous iteration, then it is useless to process it again — i.e. we may skip the above basic step —, because the information it conveys has already been added to the set of relations attached to the LHS non-terminal. Recall that the nature of this information (a set of binary relations on its attributes, expressing their mutual dependencies) is such that a single piece of information (a single dependency), once inserted, cannot be subsequently deleted (even with the covering).

The resulting algorithm, named A5 to be coherent with the notations of [DJL 84], is then algorithm A 3P modified to use the following algorithm A2SS (A2 with semantic stability)

**Algorithm A2SS** ($B$ : subgrammar) **is** :
1 **for each** $X$ **in** $B$ **do** $D(X) = \emptyset$ ; **let** $it = 0$
2 main : **repeat**
**let** $it = it + 1$ ; **let** convergence = true
3 outer : **for each** $X_0$ **in** $B$ **do**
4 **for each** $p$ **in** $P$ **such that** $p : X_0 \rightarrow X_1 X_2 ... X_{n_p}$ **do**
5 inner : **for each** $(d_1, d_2, ... d_{n_p}) \in D(X_1) \times D(X_2) \times ... \times D(X_{n_p})$ **do**
6 check : **if** $\exists\, i,\ 1 \leq i \leq n_p$, age $(d_i) \geq$ minimum-age $(it, p)$ **then**
7 **let** $d = DO(p) \cup \bigcup_{j=1}^{n_p} j.d_j$
8 **compute** $d^+$
9 **if** $d^+$ **is** circular **then** $AG$ is circular
**stop**
**endif**
10 **let** $d_0$ **be** the restriction of $d^+$ to $A(X_0)$
11 increase : **if** $d_0$ **is not** covered by $D(X_0)$ **then**
12 **let** age $(d_0)$ = current-age $(it, p)$
13 **let** $D(X_0)$ = covering $(D(X_0) \cup \{d_0\})$

```
14                                        let convergence = false
                                        endif
15                                    endif check
16                                endfor inner
17                            endfor
18                        endfor outer
19    until convergence.
```

The correctness of this algorithm is strongly dependent on the choice of functions *minimum-age* and *current-age*. We define *current-age* as follows :

$$\text{current-age}\ (it, p) = (it, p)$$

that is, the ordered pair composed of the current iteration number and production number. We also use as an order on these ages the lexicographic order on pairs :

$$(it, p) \leq (it', p') \Leftrightarrow it < it' \text{ or } (it = it' \text{ and } p \leq p')$$

Let us now state the conditions that must be fulfilled by *minimum-age* :

(i) $\text{minimum-age}\ (it,p) = (it\text{-}2,p)$ ◊

Any relation created at iteration *it*-2 on a LHS non-terminal has been processed at iteration *it* -1 on RHS non-terminals and is thus useless at iteration *it* and later.

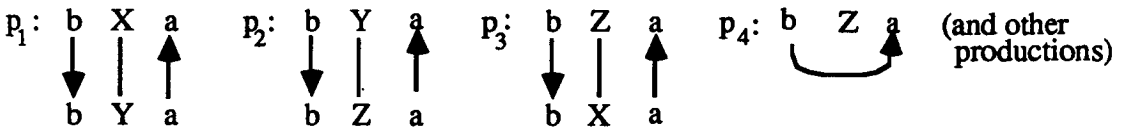
(ii) To achieve a finer condition on the ages, we must impose that the productions are always processed in the same order at every iteration in the same subgrammar (line 4). This condition is easily verified in any implementation of the algorithm : it is generally the order of textual appearance in the attribute grammar. In this case, and if we assume that the productions are ordered as they are processed, the condition becomes :

$$\text{minimum-age}\ (it, p) = (it - 1, p - 1)$$ ◊

(iii) We could state an even finer condition if we took into account the order in which the set of relations $D(X_i)$ of each RHS non-terminal is processed. However, it is difficult to insure a uniform order when the current production $p$ is recursive (i.e. $\exists\ i,\ 1 \leq i \leq n_p,\ X_0 = X_i$ ) because newly created relations on $X_0 = X_i$ modify this order ; this is especially true when covering is used, because adding a relation may delete some others. So we do not pursue this idea.

The correctness of the above definitions is obvious.

*Example 2 [Par 85] :*




The set {X, Y, Z} is an equivalence class for the grammar.

Let us assume that the non-terminals are processed in the order X, Y, Z (implying that the productions are processed in the order $p_1$, $p_2$, $p_3$, $p_4$). The non-circularity test for this subgrammar proceeds as follows, where a dot means the execution of the basic step :

without semantic stability :

| non-terminal production | X $p_1$ | Y $p_2$ | Z $p_3$ | Z $p_4$ | |
|---|---|---|---|---|---|
| iteration | | | | | |
| 1 | • | • | • | • | |
| 2 | • | • | • | | ($p_4$ not processed because of weak stability) |
| 3 | • | • | • | | |
| 4 | • | • | • | | (last iteration to check convergence) |

Total : 13 basic steps executed.

with semantic stability :

| non-terminal production | X $p_1$ | Y $p_2$ | Z $p_3$ | Z $p_4$ |
|---|---|---|---|---|
| iteration | | | | |
| 1 | • | • | • | • |
| 2 | | • | | |
| 3 | • | | | |
| 4 | | | • | |

Total : 7 basic steps executed.

Note that semantic stability may be used independently from, or in combination with, all the other improvements described in [DJL 84] or hereafter.

## 4. Non-terminals static ordering

In any circularity test, the set of dependency graphs $D(X)$ attached to each non-terminal $X \in N$ depends on the dependency graphs $D(X_i)$ for each $X_i$ in the right-hand sides of productions of which $X$ is the left-hand side, i.e. for each $X_i$ such that $X \; \Gamma \; X_i$. In order that such information propagates faster, i.e. in order to reach convergence with less iterations, it is natural to take into account the syntactic dependency relation $\Gamma$ to (statically) find an optimal order for

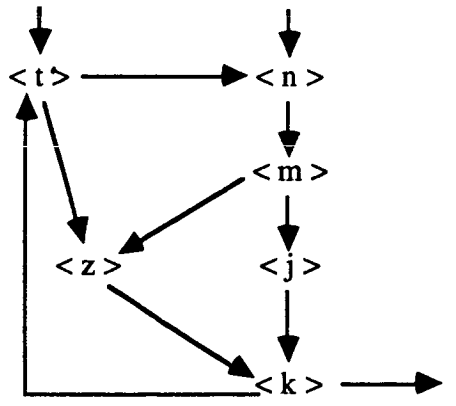
processing the non-terminals inside each equivalence class. This is achieved by topologically sorting the non-terminals in each class according to the inverse relation $\Gamma^{-1}$. However, by definition, each subgrammar is a strongly connected component of the relation $\Gamma$ (and $\Gamma^{-1}$), so it is *a priori* impossible to find a total order compatible with $\Gamma^{-1}$ ; heuristics will help make the choice.

In the same way that we defined *entry points* of a subgrammar (see section 2), we can define *output points* of a subgrammar : $X$ is an output point of class $B$ iff $\exists\, Y \in C, X\ \Gamma\ Y$ and $B \neq C$ or there exists no $Y \in N, X\ \Gamma Y$ ($X$ derives only terminal or empty productions). The first heuristic to order the non-terminals will be to order them from the output points to the entry points in each class; it is natural to do so because the former will be "lower" in derivation trees than the latter.

However this gives no indication about the relative ordering of output non-terminals of a given class. To solve this difficulty, it is also natural to take the relation $\Gamma^{-1}$ into account, as shown by the following example :

*Example 3 :*

take the following class and relation $\Gamma^{-1}$ :



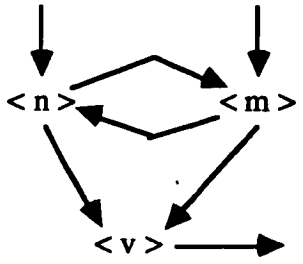

< t > and < n > are output points, < k > is the entry point.

It is obvious that < t > should precede < n > in the final ordering, such that information computed on < t > can be used on < n > in the same iteration. ◊

However, this is not sufficient either, as shown by another example :

*Example 4 :*

Class and relation $\Gamma^{-1}$ :




It is impossible to find an absolute order between < n > and < m >. ◊

In this case, and also when we encounter a similar blocking situation in the course of the topological ordering, we make an arbitrary choice.

The ordering algorithm is then as follows :

**Algorithm NT-ordering** (*B* : subgrammar) **returns ordering is** :

```
/* look for "genuine" output points in B */
1     let OP = Ø /* OP will be the set of output points in B */
2     for each X ∈ B such that
            either there exists no Y ∈ N, X Γ Y
            or there exists Y ∉ B, X Γ Y
            do add X to OP
      endfor
      restrict Γ to B
3     /* delete from OP all the non-terminals which derive directly into another output point */
      for each X ∈ OP such that ∃ Y ∈ OP, X Γ Y do
                  delete X from OP, unless this would make OP empty
                        /* arbitrary choice in case of blocking situation */
      endfor
4     /* order elements of OP arbitrarily */
      let i = 1
      for each X ∈ OP do
            give X rank i
            let i = i + 1
            delete X from Γ, along with all the edges reaching it and departing from it
      endfor
5     /* topologically order the other members of B */
      while i ≤ |B| do
            if there exists X ∈ B with no edges departing from it in Γ
                  /* simple topological sort */
                  give X rank i
                  let i = i + 1
                  delete X from Γ, along with all the edges reaching it
            else /* hard case : make an arbitrary choice */
                  determine NP /* no predecessors */, a strongly connected component of Γ
                        with no edges departing from it
                  choose an arbitrary X in NP
                  give X rank i
```

```
                let i = i + 1
                delete X from Γ, along with all edges reaching or departing from it
            endif
        endwhile
end NT-ordering.
```

This algorithm is obviously polynomial in the size of the grammar.

*Example 3 (continued) :*

A possible ordering derived by the above algorithm would be < t >, < n >, < m >, < j >, < z >, < k >. ◊

*Example 1 (continued) :*

In case of class $B_6$, a possible ordering would be $A_{15}$, $A_{11}$, $A_{14}$, $A_{10}$, $A_7$, $A_{16}$. This example is interesting because the last non-terminal in the ordering, $A_{16}$, is not an entry point in class $B_6$. ◊

To integrate this ordering in the complete circularity test algorithm, we need to modify algorithm A2 by applying the above algorithm in the initialization part (line 1) and by processing the non-terminals (line 3) in the corresponding order. The final algorithm, which we shall name A6, is then A3P modified to use the above modified A2 (A6 is not shown).

The advantages of this non-terminals static ordering is that, as expected, information propagates faster, as shown by the important reduction in the number of iterations needed to reach convergence (see section 5). Note that determining a "better" ordering for traversing a graph to propagate information is a well-known technique in other areas of computer science — e.g. in data flow analysis, where it is called "reasonable node listing" —, however its efficiency is particularly important in the present application to non-circularity tests.

Also note that it is possible to use this static ordering independently from Chebotar's partitioning, that is on the whole grammar :

*Example 1 (continued) :*

Assuming that $A_{12}$ and $A_{13}$ derive terminal productions, a possible ordering of the complete set of non-terminals is $A_{12}$, $A_{13}$, $A_8$, $A_5$, $A_9$, $A_6$, $A_{15}$, $A_{11}$, $A_{14}$, $A_{10}$, $A_7$, $A_{16}$, $A_4$, $A_3$,$A_2$,$A_1$. ◊

However their combination is very effective, because the advantages of partitioning are fully retained in any case : gain in time because each iteration processes fewer non-terminals,

gain in space because we can forget useless relations. Non-terminals static ordering adds up on these advantages by decreasing the number of iterations in each subgrammar.

Combining static ordering and semantic stability causes no problem because their actions are orthogonal, as can be seen from algorithm A2 : static ordering acts on line 3, by giving a different order for processing the non-terminals, while semantic ordering acts on line 5 by restricting the number of graphs combinations which should be processed. But one could think that, since information propagates faster with static ordering, a very large majority of these graphs combinations are useful and cannot be ignored by semantic stability. As the results of section 5 show, this is partly true ; however it appears that semantic stability is not completely subsumed by static ordering, and that their combination is more efficient than each of them separately.

## 5. Experimental Results

We have implemented all these algorithms on the Multics system at INRIA, and tried them on some real examples. This section presents some statistics we gathered from their execution. We will use the following notations, mostly borrowed from [DJL 84] :

For each attribute grammar :

| | | |
|---|---|---|
| $nt$ | = | number of non-terminals |
| $pr$ | = | number of productions |
| $mnt$ | = | maximum number of non-terminals in right sides of productions |
| $\alpha$ | = | total number of attributes |
| $mna$ | = | maximum number of attributes per non-terminal |
| $k$ | = | number of subgrammars according to Chebotar's partitioning |

For each algorithm :

| | | |
|---|---|---|
| $d_{max}$ | = | maximum number of relations in any $D(X)$ at any time |
| $d_{av}$ | = | average number of relations in any $D(X)$ at any time |
| $R_{max}$ | = | maximum number of co-resident relations, i.e. |

$$\max_{time} \left( \sum_{X} | D(X) | \right)$$

| | | |
|---|---|---|
| $it_{max}$ | = | maximum number of iterations on each subgrammar |
| $it_{av}$ | = | weighted average number of iterations on each subgrammar, i.e. |

$$\frac{\sum_{j=1}^{k} it_j \times pr_j}{pr}$$

| | | |
|---|---|---|
| $tcl$ | = | total number of transitive closures (basic steps) computed |
| $cpu$ | = | CPU time |

All these variables are the same as in [DJL 84] ; we added another one, namely $tcl_{circ}$, which is the number of transitive closures computed for actually testing non-circularity. Indeed, the algorithms we have implemented differ from the ones which have been presented in [DJL 84] or here in that the circularity of each composite dependency relation ($d^+$ in A2, section 2) is not tested in the basic step (line 8), but rather in a separate pass, after the set of relations of each non-terminal has reached convergence (writing the algorithm is left to the reader). Of course, this increases the number of transitive closures which must be computed, but, provided that the grammar is non-circular, this saves many more circularity tests which are almost as expensive (complexity $mna^2$). Since most of the attribute grammars are expected to be non-circular, doing so results in a net gain. This $tcl_{circ}$ figure is interesting since, being the number of final graph combinations, it is a very good measure of the "semantic complexity" of a given attribute grammar.

Now a word on our examples :

- *simproc* is a simple semantic checker and translator for a toy language with integers, arrays and recursive procedures ;
- *asm* is a cross-assembler generating hexadecimal code for an MC 6809 micro-processor ;
- *simula* is a compiler for the language Simula, borrowed from Fang [Fan 72], but modified to make it non-circular ;
- *pl1_c* is a translator from a subset of PL/1 to C [Maz 86] ;
- *pascal* is a full Pascal to P-code compiler.

All examples except *pl1_c* are nearly, but not quite, the same as those described in [DJL 84] ; this explains the slight differences between the measures on A4 in [DJL 84] and here. Also, since we wrote [DJL 84] , our hardware was improved, which accounts for the importance difference on the CPU time for A4.

We added statistics for strong non-circularity tests, both as a comparison point and because all our examples are strongly non-circular.

The rough results are presented in Table 1.

The first comment which comes to mind when reading this table is that our improvements are *always* effective ; the gain on the number of transitive closures is even very impressive for large examples (about 3.5 times for *pascal*).

| | | simproc | asm | simula | pl1_c | pascal |
|---|---|---|---|---|---|---|
| | *nt* | 9 | 58 | 126 | 139 | 115 |
| | *pr* | 20 | 216 | 244 | 376 | 216 |
| | *mnt* | 5 | 3 | 6 | 8 | 6 |
| | $\alpha$ | 9 | 22 | 37 | 48 | 60 |
| | *mna* | 7 | 10 | 17 | 16 | 15 |
| | *k* | 4 | 57 | 58 | 92 | 58 |
| | $d_{max}$ | 1 | 3 | 5 | 4 | 8 |
| | $d_{av}$ | 0.81 | 0.62 | 0.96 | 0.94 | 1.25 |
| | $R_{max}$ | 4 | 18 | 79 | 52 | 66 |
| | $it_{max}$ | 4 | 3 | 13 | 12 | 21 |
| | $it_{av}$ | 3.20 | 1.11 | 7.70 | 4.20 | 7.12 |
| A4 | *tcl* | 65 | 562 | 2846 | 1871 | 7735 |
| | *cpu* | 5.80 | 20.4 | 98.1 | 121. | 523. |
| A5 | *tcl* | 55 | 555 | 1675 | 1128 | 3258 |
| | *cpu* | 5.72 | 19.9 | 74.6 | 101. | 252. |
| | $d_{max}$ | 2 | 3 | 5 | 4 | 7 |
| | $d_{av}$ | 0.88 | 0.62 | 1.05 | 1.00 | 1.20 |
| | $R_{max}$ | 5 | 18 | 94 | 52 | 65 |
| | $it_{max}$ | 4 | 3 | 7 | 6 | 8 |
| | $it_{av}$ | 2.85 | 1.07 | 4.28 | 2.27 | 3.41 |
| A6 | *tcl* | 62 | 555 | 1974 | 1220 | 3170 |
| | *cpu* | 5.72 | 19.8 | 79.6 | 103. | 228. |
| A7 | *tcl* | 55 | 550 | 1511 | 995 | 2228 |
| | *cpu* | 5.70 | 19.8 | 70.7 | 96.1 | 182. |
| | $tcl_{circ}$ | 20 | 269 | 272 | 389 | 457 |
| SNC | *tcl* | 58 | 274 | 640 | 570 | 779 |
| | *cpu* | 0.35 | 2.32 | 15. | 17. | 23. |

**Table 1 : Practical Comparison of Non-Circularity Test Algorithms**

It is impossible to prove formally that our algorithms are optimal, and we won't claim that either. Let us however examine the processing of a "simple" subgrammar : we need (at least) one iteration for constructing the relations, one iteration to check that convergence is reached (this iteration may be suppressed by weak stability) and, in our scheme, one iteration for actually testing non-circularity ; thus, for a "simple" grammar, the total number of transitive closures is of the order of three times the number of transitive closures needed for testing non-circularity. With our improvements we reach this behavior for three of our examples, including one large grammar *(pl1_c)* ; this is, in our opinion, an indication that A7 is near-optimal.

Let us take a closer look at the results. First, semantic stability does not change the order of the basic computations ; it only omits to make some computations which are anyway redundant ; this explains why it has no effect on the number of iterations ($it_{max}$, $it_{av}$) or on the space needed to store the relations ($d_{max}$, $d_{av}$, $R_{max}$) (see A5 vs. A4 and A7 vs. A6). On the other hand, as foreseen, non-terminals static ordering strongly decreases the number of iterations necessary to reach convergence, an evidence that information propagates faster. Its effect on space is variable from a grammar to another. The increase of $R_{max}$ for *simula* (from 79 to 94) is to be interpreted as follows : since the same quantity of information (the sets of dependency relations) must be gathered in a smaller number of steps (iterations), it is rather normal that the amount of information which must be stored at a given time increases. So this is not bothering. However for *pascal* we notice a slight decrease of $R_{max}$ (from 66 to 65) ; this is due to covering : with A6 one of the relations of one non-terminal is covered by another one before the creation of a third one, while with A4 the first and the third relation relations are created before the first being covered by the second. Everything being considered, we may say that static ordering has no strong effect on space consumption ; a formal analysis, if it were possible, would certainly show the same space complexity.

The fact that the SNC test for *simproc* is more expensive, in terms of transitive closures, than our best NC test A7 may be explained as follows : on one hand, the only non-terminal having two relations in the NC test is directly recursive ; on the other hand, the relation attached to it in the SNC test is the sum of those two NC relations, and is thus more complicated. In the NC test, those two relations are processed in the same iteration, while another iteration is needed in the SNC test. This explains the unusual difference. However, this not bothering because *simproc* is so simple an example that it is nearly meaningless.

## 6. Conclusion

We presented two new improvements to be applied to algorithms testing non-circularity. Semantic stability avoids a large number of redundant computations. Non-terminals static ordering finds a better processing ordering which makes information propagate faster. Both of them allow significant gains in execution time, which make circularity tests feasible in practice. Their implementation is easy, and the combination of these two improvements with the three ones presented in [DJL 84] seems to be near-optimal, so that future gains can only be realized through careful implementation.

These improvements, especially static ordering, can be applied to other algorithms processing grammars and attributes, including strong non-circularity tests [Par 85] and transformation of strongly non-circular attribute grammars into L-ordered ones [Par 88].

**Acknowledgements** : thanks go to Bernard Lorho and Pierre Deransart for useful comments on a previous draft of this paper.

## 7. References

[Che 81] Chebotar, K.S. : "Some Modifications of Knuth's Algorithm for Verifying Cyclicity of Attribute Grammars", *Progr. Comput. Soft.* **7** (1), pp. 58-61 (1981).

[Che 85] Chebotar, K.S. : "Cyclicity in Attribute Grammars : Practical Approach", French-Soviet Colloquium on Computer Science, Tallinn, USSR (1985). Unpublished communication.

[DJL 84] Deransart P., Jourdan M., & Lorho B. : "Speeding up Circularity Tests for Attribute Grammars". *Acta Informatica* **21**, pp. 375-391 (1984).

[DJL 85] Deransart P., Jourdan M., & Lorho B. : "A Survey on Attribute Grammars, Part III : Classified Bibliography", Rapport RR-417, INRIA, Rocquencourt, France (1985).

[Fan 72] Fang I. : "FOLDS, a Declarative Formal Language Definition System", PhD thesis, report STAN-CS-72-329, Computer Science Dept., Stanford University, Stanford, CA (1972).

[JOR 75] Jazayeri M., Ogden W.F., & Rounds W.C. : "The Intrinsically Exponential Complexity of the Circularity Problem for Attribute Grammars", *CACM* **18** (12), pp. 679-706 (1975).

[Knu 68] Knuth D.E. : "Semantics of Context Free Languages", *Math. Systems Theory* **2** (2), pp. 127-145 (1968).

[Knu 71] Knuth D.E. : "Semantics of Context Free Languages : Correction", *Math. Systems Theory* **5** (1), pp. 95-96 (1971).

[LP 75] Lorho B., & Pair C. : "Algorithms for Checking the Consistency of Attribute Grammars", in *Proving and Improving Programs*, Huet G., & Kahn G. (eds.), Colloque IRIA, Arc-et-Sénans, France, pp. 29-54 (1975).

[Maz 86] Mazaud M. : "Un Traducteur de PL/1 vers C", rapport technique RT-65, INRIA, Rocquencourt, France (1986).

[Par 85] Parigot D. : "Système Interactif de Trace des Circularités dans une Grammaire Attribuée et Optimisation du Test de Non-Circularité", rapport de DEA, Université de Paris XI, Orsay (1985).

[Par 88] Parigot D. : "Practical Transformation of Absolutely Non-Circular Attribute Grammars into L-ordered ones", submitted for publication (1988).

[RS 82] Räihä K.J., & Saarinen M. : "Testing Attribute Grammars for Circularity", *Acta Informatica* **17**, pp. 185-192 (1982).